AFRL-IF-RS-TR-2005-119
Final Technical Report
April 2005

INTRUSION TOLERANCE BY UNPREDICTABLE ADAPTATION (ITUA)

BBNT Solutions, LLC

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. K446

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

STINFO FINAL REPORT

AFRL-IF-RS-TR-2005-119 has been reviewed and is approved for publication

APPROVED: /s/

PATRICK M. HURLEY
Project Engineer

FOR THE DIRECTOR: /s/

WARREN H. DEBANY, JR., Technical Advisor
Information Grid Division
Information Directorate

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 074-0188*

1. AGENCY USE ONLY (Leave blank)	**2. REPORT DATE** APRIL 2005	**3. REPORT TYPE AND DATES COVERED** Final Jul 00 – Jan 04

4. TITLE AND SUBTITLE INTRUSION TOLERANCE BY UNPREDICTABLE ADAPTATION (ITUA)	**5. FUNDING NUMBERS** C - F30602-00-C-0172 PE - 63760E PR - K446 TA - 15 WU - A1
6. AUTHOR(S) Partha Pal, Michael Atighetchi, Chris Jones, Idit Keidar, David Levin, Joseph Loyall, Paul Rubel, Richard Schantz, Ronald Watro and Franklin Webber	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) BBNT Solutions LLC 10 Moulton Street Cambridge Massachusetts 02138	**8. PERFORMING ORGANIZATION REPORT NUMBER** N/A
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Defense Advanced Research Projects Agency 3701 North Fairfax Drive Arlington Virginia 22203-1714 AFRL/IFGA 525 Brooks Road Rome New York 13441-4505	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER** AFRL-IF-RS-TR-2005-119

11. SUPPLEMENTARY NOTES

AFRL Project Engineer: Patrick M. Hurley/IFGA/(315) 330-3624/ Patrick.Hurley@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.	**12b. DISTRIBUTION CODE**

13. ABSTRACT ***(Maximum 200 Words)***

The ITUA project began with the research goals of using Byzantine fault-tolerant protocols to coordinate adaptation to attacks and exploring the use of unpredictability in adaptive responses to confuse and delay the attacker. These main factors distinguish the ITUA approach; 1) dynamic adaptation - intrusions cause charges in the system, and a survivable system must cope with these changes, 2) a defense-enabled application that has an application and mission specific defense strategy, and 3) defense enabling builds the defense in middleware, intermediate between the application and the networks and operating systems on which the applications run. The ITUA approach was to respond and adapt to the effects of a malicious attack while it is in progress. If the defense-enabled application continues correct processing in spite of the attached, the defense was considered to be successful. The report reviews the motivation for the project and its historical context, then summarizes the ITUA research goals, progress toward achieving those goals, and lessons learned from the research. The report offers some plans for future research and draws a conclusion.

14. SUBJECT TERMS Intrusion Tolerance by Unpredictable Adaption, Defense-Enabled Applications, Adaptive Defenses in Middleware, Dynamic Adaption, Unpredictability in Adaptive Responses	**15. NUMBER OF PAGES** 44
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	**18. SECURITY CLASSIFICATION OF THIS PAGE** UNCLASSIFIED	**19. SECURITY CLASSIFICATION OF ABSTRACT** UNCLASSIFIED	**20. LIMITATION OF ABSTRACT** UL

Contents

List of Figures

1 Identification

The Intrusion Tolerance by Unpredictable Adaptation (ITUA) project was a joint effort by BBN Technologies, the University of Illinois, the University of Maryland, and the Boeing Company. The work was supported by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL) under contract F30602-00-C-0172 beginning in July 2000.

This document is the Final Report for ITUA. It reviews the motivation for the project and its historical context, then summarizes the ITUA research goals, progress toward achieving those goals, and lessons learned from the research. When research results are mentioned, the reader is directed to other documents that provide more detail. A list is given of specific achievements and documents created by the ITUA project. Finally, the Report offers some plans for future research and draws a conclusion.

This document is delivered in fulfillment of CDRL A003 and in partial fulfillment of CDRL A011 of the contract.

2 Project Motivation

Malicious attacks against computer systems are becoming more common and more damaging. A malicious attack against computer system X is a sequence of aunauthorized actions taken by a user (or a group of users) of some computer system Y to alter or deny data processing in X. If X and Y are the same, the user is called an insider (i.e., he is authorized to use X but not to attack it); if they are different, he is called an outsider (e.g., he attacks X over the Internet from Y). Malicious attacks are becoming more common because computers are becoming more interconnected. Malicious attacks are becoming more damaging because computers are increasingly relied on to provide essential data and services, so unauthorized changes to these data and services are more harmful and noticeable, and less tolerable.

A secure computer system protects against malicious attacks by making such attacks very difficult or impossible to carry out. Unfortunately, building a completely secure computer system is now known to be exceedingly difficult, an ideal that is very costly during both development and maintenance, and that involves significant burdens for users. Practical alternatives are needed.

The goal of the ITUA project was to give software applications an increased resistance against malicious attack even when they run in an environment that is not completely secured. We call any such application “defense-enabled”. Note that defense enabling is less ambitious than building a secure system: rather than protect the entire system, defense enabling concentrates on the survival and integrity of essential applications, possibly sacrificing other parts of the system to the attacker. Defense enabling also gives priority to some security properties over others: we are much more concerned with defending the availability and integrity of an application’s services and data than their confidentiality. Note that this priority is implicit in our definition of “malicious attack”, which does not mention the possibility that the attacker might steal confidential data. Also note from the definition that defense enabling is resistance to *unauthorized actions* by insiders and/or outsiders. An attack by an insider who takes only *authorized actions*[1] is beyond the scope of the project.

The ITUA approach was, in brief, to respond and adapt to the effects of a malicious attack while it is in progress. If the defense-enabled application continues correct processing in spite of the attack, we consider the defense to be successful. Even if the attacker eventually is able to damage the application, the defense will have been successful if it allowed time for system administrators to detect the attack and respond to it, perhaps by physically blocking the attacker’s continued access to the system. Any technique for extending an application’s life while under malicious attack is potentially part of defense enabling.

3 Historical Context

Defense enabling is representative of a relatively recent trend in computer security, often called *survivability* or 3rd generation security. The 3 generations can be summarized as follows:

1. 1st generation security aimed to protect systems completely with software and hardware mechanisms that could not be circumvented [19]. This approach has proved to be quite costly and inflexible.

2. 2nd generation security acknowledged that implementations of 1st generation protection are imperfect, allowing attackers to circumvent security mechanisms. The 2nd generation aimed to detect attacks, allowing system operators to respond [8].

3. 3rd generation security uses both 1st and 2nd generation techniques but adds automated defenses that respond to an attack. These defenses can respond

[1]This is sometimes called an “insider attack”.

to the effect of attacks even when the attacks themselves are not detected with 2nd generation techniques.

The 3rd generation has proved to be necessary in modern computer systems, which typically rely on "commercial off-the-shelf" (COTS) components whose security characteristics are not known and that interconnect and interoperate with many other systems. The use of insecure COTS means that the 1st generation approach is impossible. The openness of these systems results in the constant discovery and use of new attacks. When 2nd generation techniques fail to detect these new attacks, 3rd generation techniques are necessary [2].

Several factors distinguish the ITUA approach to survivability from others. First, dynamic adaptation is a key theme of our approach. Intrusions cause changes in the system, and a survivable system must cope with these changes. As a consequence, defense-enabled applications must be very agile and will make use of the flexibility possible in today's dynamic, networked environments. Second, a defense-enabled application has a defense strategy that is typically application- and mission-specific. Such strategies complement and go beyond traditional approaches to security in which protection mechanisms are typically not aware of the applications they aim to protect. Third, defense enabling builds the defense in middleware, intermediate between the application and the networks and operating systems on which the application runs. Recent developments in middleware allow us to develop adaptive defenses quickly [15, 10, 6]. Advanced middleware allows us to base these defenses on information from layers both above and below, i.e., from the application and from the infrastructure, as well as allowing us to control and dynamically adjust aspects of each layer. Defense strategies implemented in middleware can be reused relatively easily in the context of other applications because they are only loosely coupled to the application.

With ITUA, we also extended our own previous work on defense enabling. The APOD project ("Applications that Participate in their Own Defense") had similar, but more limited, goals. By subjecting a defense-enabled application to an attack by a Red Team, we were able to show during APOD that dynamic adaptation could slow the progress of even a sophisticated attacker [12]. ITUA improved upon that work in several ways, as discussed in the next section.

Near the end of the ITUA project, we began the OASIS Dem/Val project in which we applied some of the ITUA ideas to the defense of an experimental military system. That system, a Joint Battlespace Infosphere (JBI) [18], is intended for command and control and would be an attractive target for attack. As of this writing, our work to create a defense-enabled JBI, named "DPASA", is still in progress. The design and validation of DPASA, although incomplete, has already influenced some of the conclusions we will draw about ITUA in this document.

4 Project Goals

The ITUA project began with the following research goals:

- Use Byzantine fault-tolerant protocols to coordinate adaptation to attacks.
- Explore the use of unpredictability in adaptive responses to confuse and delay the attacker.

Byzantine fault-tolerant protocols were originally invented, two decades ago, for use in ultra-reliable avionics [20]. Byzantine fault tolerance depends on redundancy to isolate and hide the effects of (i.e., "mask") failures that cause some components to behave in arbitrarily bad ways.

Malicious attack could cause such arbitrarily bad behavior, and so the idea of using Byzantine fault tolerance to mask the effects of attacks was widely recognized. Arbitrarily bad behavior could be caused, for example, if an attacker defeats the protection mechanisms on a host H to gain privileges on H, then uses that privilege to corrupt an application component running on H. With enough redundant components on other hosts, a Byzantine fault-tolerant protocol could mask bad decisions by the corrupt component on H, and other components might even agree to ignore H altogether.

Byzantine fault tolerance has not been widely used in distributed systems, mainly because it is expensive. Protocols to tolerate arbitrarily malicious behavior need more redundancy and are slower than protocols that tolerate only "crashes". Our own APOD work explicitly avoided using Byzantine fault tolerance, with the rationale that few attackers have both the skill and motivation to create malicious behavior beyond crashes, and so merely tolerating crashes would raise the bar against most attacks.

Whether Byzantine corruption is common or not, however, the threat is real. Many published vulnerabilities in operating systems (e.g., see [3]) allow remote attackers to execute arbitrary code with administrator privilege. Some attackers will exploit these vulnerabilities, then execute code custom-built to attack the target application. This kind of attack is especially likely if the attacker's goal is to damage the application's integrity rather than to deny service. It is this kind of attack that ITUA was designed to counter (as well as less sophisticated attacks!).

Our work to make Byzantine fault tolerance useful for defense enabling was quite successful and is summarized in Section 5.5.

Most attacks, especially those launched by an outsider, will follow a plan that reflects the attacker's knowledge of potential vulnerabilities. In its simplest form, the plan for a corruption-inducing attack is:

1. find a vulnerable host that can be commandeered remotely to give the attacker administrative privilege;

2. execute preplanned attack code to corrupt the application running on that host.

Either of these steps can involve multiple actions, and some of the actions may be scripted (i.e. run in an automated manner). A more complicated plan will take the form of an attack tree, in which the branches represent alternatives the attacker will try depending on how the defense reacts.

It is our thesis that unpredictability, in the form of reactive indeterminacy or proactive changes in the system, can be used to the advantage of the defense. Proactive changes in the system may quickly make the attacker's plan obsolete. Reactive indeterminacy may break the attacker's scripts, complicate the attack plan or scare the attacker off. At the very least, reactive indeterminacy forces the attacker to consider more possibilities at branches in the attack tree, any of which might arise in a particular attack run. Even if there is only a small number of such possibilities, the uncertainty about which outcome the attacker will face makes the plan and its scripting more complicated, effectively slowing down the attack.

In ITUA, we used unpredictability as a technique to enhance the effectiveness of our adaptive responses. We built middleware support for adding randomness to an application's adaptive behavior, to the low-level rapid response mechanisms and to the management of redundant resources. We also started to evaluate and quantify the impact of unpredictability. These issues are summarized in Section 5.6.

ITUA was the first project to explore systematic use of unpredictability in conjunction with adaption. Other 3rd generation security and survivability projects have now also adopted unpredictability to augment their defensive capability. Below are some examples:

- Countering Traffic Analysis in IPSec
  - Work in this project aims to use unpredictability to hinder information gathering by traffic analysis. For instance, even on an encrypted channel, the last hop normally reveals the the endpoint, which along with other observations such as volume, time of day, current world events (war or worm attack) could yield potentially valuable information for the attacker. By generating additional and random hops after the real recipient, traffic analysis becomes harder.

- Distributed Authentication for Mobile Internet
  - This project used the uncertainty prevalent in a mobile Internet to develop a distributed authentication mechanism. In this context, N different packets from source A to source B may be forwarded in N different ways, because source and/or destination can be mobile. Distributed authentication will mean that these N packets may be encrypted (signed) in N different ways as well. A distributed authentication mechanism is claimed to be more resilient than its less dynamic and more centralized alternatives.
- Probabilistic Routing/Adaptive Multipath Routing
  - Multiple projects are exploring a set of new protocols that address the vulnerabilities of static routing (that the path from source A to destination B is fairly static and predictable), and tolerate accidental and malicious failures in control (routing) and data (transport). Anonymizers, overlays, dynamic selection of routes are examples where uncertainty and dynamism are used for protection and tolerance.

The ITUA project also concentrated on several subordinate goals, necessary to evaluate our research on defense enabling:

- Create an architecture for managing defensive adaptation. (See Section 5.1 for a list of architectural issues.)
- Define policies for coordinated, global, adaptation. (See Section 5.3 for a list of global adaptation issues.)
- Define policies for rapid, local, response to attacks. (See Section 5.4 for a list of local response issues.)
- Implement a prototype. (See Section 5.8 for lessons learned during implementation.)
- Demonstrate technology transfer feasibility by defense-enabling an application of military interest. (See Section 5.7 for a discussion of how ITUA was applied to IEIST, a system developed by our team member, Boeing.)

Throughout the project, we used mathematical models to understand and validate the effectiveness of ITUA defenses. Validation became such an essential part of the project that DARPA awarded us several extensions to explore validation techniques:

- In early 2002, we were tasked to apply probabilistic validation techniques to ITUA's intrusion tolerance. The results from this task are reported in detail in a separate project document, "Validating Intrusion Tolerance Using Mathematical Models", hereafter referred to simply as the "Validation Report".
- In late 2002, we were tasked to devise techniques to measure and quantify assurance of large, distributed information systems and to develop a 5- to 10-year research roadmap. We presented our research roadmap in this area at a workshop in Oglebay. Under this award, we also executed performance and adversarial testing of ITUA technologies.

Our validation results are intertwined with progress toward the other goals and will be discussed throughout the next section. In particular, though, see Section 5.2 for assumptions we made about potential attackers and see Section 5.9 for a discussion of issues that arose during the validation process itself.

In March 2002, the ITUA team was invited by DARPA to join 5 other OASIS performers to participate in an workshop about integrating OASIS technologies into the design of a survivable server. The workshop considered several military server applications and selected a security management service from Army CECOM. This choice was the beginning of the SAMS project, in which technologies from ITUA (BBN and UI), HACQIT (Teknowldege), KARMA (Draper), ITSI (SCC), JBET (NAI) and VPNShield (ATC) were integrated to defend CECOM's application at various layers of its tactical network. This project was fast-paced, risky and highly visible, and demonstrated that OASIS research could be usefully integrated into a real military application that came with its own requirements, including constraints on the size of, and space allocated for, hardware. The SAMS project team developed a prototype, which was demonstrated both in the development lab and at CECOM.

5 Research Progress and Issues

This section lists the main research questions that we addressed during the ITUA project. For each question, we summarize our answer, and if necessary, direct the reader to other documents that provide more detail.

5.1 Architecture

The ITUA architecture organizes the defense of an application into several layers:

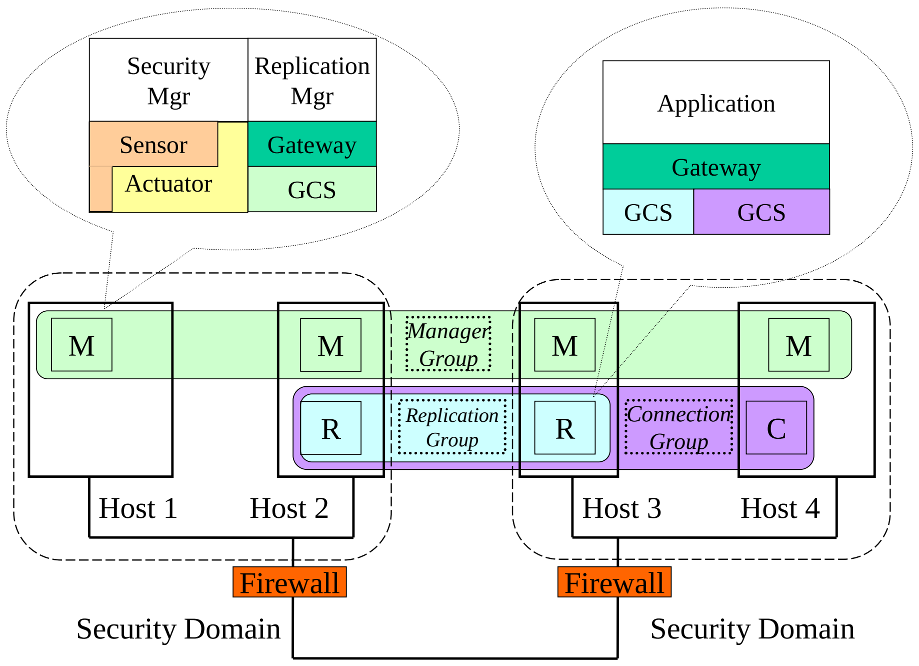

Figure 1: ITUA Architecture

- **Application Components:** Critical application components are replicated, and each *group* of replicas is coordinated using a Byzantine fault-tolerant protocol. The coordination implements the well-known State Machine Approach for providing fault-tolerant services [17].

- **Network Hosts:** The defense of each network host is controlled by an ITUA *manager* on that host. A defense strategy for the whole network results from coordinating the set of all ITUA managers on all hosts using a Byzantine fault-tolerant protocol to reach consensus [1]. The set of managers will decide, for example, when and where to create new application component replicas.

- **Security Domains:** Each host resides in a unique security domain. The boundaries between security domains are designed to be relatively hard for an attacker to cross. Different domains use different platforms and different platform configurations so that an attack that works in one domain is likely to fail in another. Firewalls constrain the traffic that can pass between domains. Application component replicas are placed in different domains so that an attacker must cross domain boundaries to corrupt more than one replica in any group.

A pictorial summary of the ITUA architecture can be seen in Figure 1, in which a defense-enabled client-server application is distributed over two security domains. The figure shows the client, marked by C, unreplicated (perhaps because it is a user interface), running on Host 4 in the right-hand security domain. The server has two replicas, each marked by R, running on Hosts 2 and 3, one in each domain. Each host runs a manager, marked by M. The bubbles at the top of the figure show some structure within each process. An application process interacts with replication groups via a Gateway, which uses a Group Communication System (GCS). A manager comprises a security manager component, which monitors sensors and controls actuators on the host, and a replication manager component, which controls the structure of replication groups. Both manager components use the GCS via the Gateway to coordinate with the other managers. More detail on the ITUA architecture can be found in [13].

The key architectural questions we addressed in the ITUA project are the following:

- **Should managers be replicas?**

  Coordinating the defense is made simpler if the managers agree on some decisions. For example, if processes in one domain behave badly, the defense could declare that domain to be "bad" and quarantine it. If managers do not agree, however, on which domains are bad, the defense may effectively partition the network into enclaves, each accepted by only a fraction of the managers. The easiest way to avoid this problem is for all non-corrupt managers to agree on which domains should be quarantined. Other management decisions similarly depend on agreement.

  The non-corrupt managers would agree, by design, if all managers were replicas and maintained the same state. The ITUA design replicates critical application processes, so the managers could be replicated in the same way. However, we decided against using replicated managers for two reasons:

  1. Managers that are not replicated can act autonomously and this fact adds flexibility, decentralization, and security to the design. In fact, the ITUA managers do not need to agree on every decision but only on some, so they do not need to share and agree on their entire state, only on parts of it. For example, the manager on host H will maintain data about intrusion detection alerts on H and may take local actions as a result. This local data does not need to be replicated at other managers.

     The more management data can be kept local, the more secure the design. Data that is replicated at every manager may be revealed to an attacker once even a single manager becomes corrupt. Local data, in contrast, will be compromised only when the local manager is successfully attacked.

2. Agreement protocols are expensive and so should be used sparingly, especially in a large system. The set of ITUA managers is likely to be large, and agreement protocols tend to grow much more expensive when used by large groups. Not only are distributed systems getting larger as networking costs fall, but we believe some ITUA defenses will work better in a large network. For example, a large, heterogeneous network gives the defense more options for moving an application's computation away from corrupt security domains. Therefore, the cost of using agreement among managers should be minimized by using agreement protocols only when necessary rather than to maintain replication.

For these two reasons, we decided that managers should operate autonomously for most decisions and demand agreement only for decisions in which it is essential.

- **Should managers form a hierarchy?**

  Arranging the ITUA managers into a hierarchy is one way in which the costs of agreement could be reduced. For example, suppose the root of the manager hierarchy consisted of 4 managers, running a Byzantine agreement protocol to tolerate a single corrupt manager. Decisions made at the root would be passed along to subordinate managers at the branches and leaves. The cost of agreement in this hierarchy would be only the cost for a group of size 4.

  Unfortunately, a hierarchy has the big disadvantage that its root is a far more valuable target for attack than any leaf. In the example, corrupting only two root managers would lead to unmaskable effects and to corruption of the entire defense. Even if the root nodes were changed unpredictably from time to time this weakness would still exist.

  On the other hand, the ITUA design groups hosts (and their managers) into security domains and this grouping suggests a shallow, two-layered hierarchy. Because attacks within a security domain are likely to be easier than attacks between domains, corruption of one manager within a domain may lead relatively quickly to corruption of all other managers in that domain. The security domains suggest a hierarchy in which some managers in domain D are subordinate to other, primary, managers in D. Subordinate managers can be the leaves of the hierarchy while the primary managers form the root.

For these reasons, we decided that managers should form a hierarchy with at most two layers. The root layer would consist of at least one manager from every domain. The leaf layer would consist of all other managers. Putting more managers in the root layer would add redundancy and make the defense more robust, while increasing the cost of agreement among the managers.

- **Should security domains contain many hosts or few?**

  In the Validation Report, Chapter 3, Sections 4.1 and 4.2, we studied how varying the numbers of domains and hosts will affect survivability (as defined in terms of the application's reliability and availability). The results were:

  - If the number of hosts is fixed, survivability increases with more domains.
  - If the number of domains is fixed, survivability is largely unaffected by increasing the number of hosts.

  Both results depend ultimately on the assumption that attacking within a domain is relatively easy, so adding hosts to a domain doesn't add much "space" for the application to run replicas. We concluded that a good design for survivability will have as many domains as possible and relatively few hosts per domain.

  Increasing the number of domains in a design, however, is relatively expensive (compared to the cost of adding hosts). A new security domain gains much of its resistance to attack by its uniqueness: different platforms, different configurations of the same platform, and local administration of platforms and configurations. Each of these factors costs money. For example, our DPASA design has only 4 security domains, the minimum number needed to tolerate Byzantine corruption of a single domain. Yet finding 4 different platforms that will run all the software we need has been difficult.

  So the trade-off is: use more domains for survivability; use fewer domains to cut costs.

- **Can managers change the set of security domains dynamically?**

  Security domains can be added or removed by system administrators when the system is initially configured. But the ITUA managers themselves could decide to add or remove domains at runtime, just as they decide whether to add or remove replicas of application components. Does the ITUA architecture support this?

  We decided that the managers will be able to *remove* a security domain by quarantining it, cutting off all interaction between that domain and the rest of the system, but that new security domains will be *added* only by human intervention. Consider the difficulties of adding a new domain at runtime. First, no manager in the system has any reason to trust a new domain D unless it has some prior awareness of D, i.e., D was, in effect, already part of the system. Second, a manager is ITUA's "presence" in a domain, allowing ITUA to start other processes in that domain securely. To start a trustworthy manager in D would require a trustworthy meta-manager in D, which a new domain also lacks. For these reasons, the only way to add a new domain in ITUA is by human fiat.

5.2 Attacker Model

In order to decide how to use the ITUA architecture for defense, we needed to decide what kind of attacks ITUA would defend against. This decision required us to make assumptions about the attacker: what he can do, and what he is unlikely to be able to do. These assumptions, once they are expressed clearly, can be judged for their reasonableness by comparing with actual attacks on distributed systems.

We assumed that an attacker would proceed by collecting privileges. In each security domain, the attacker would exploit some vulnerability to gain privilege on some host in that domain. Using that privilege, he would aim to exploit other vulnerabilities, either to increase his privilege on the same host or to attack another.

For example, the first step would be to attack a host H remotely, exploiting a flaw in some network application to gain the privilege used by that application, perhaps of some user U of H. The next step might be to attack host H again, this time acting locally as user U, exploiting a flaw in some operating system component to gain privileges used by the operating system. On most hosts, the attacker would then have enough privilege on H to kill or corrupt any process running on H. The attacker could then proceed to attack other hosts and other domains.

We assumed that an attacker might be prepared to corrupt some processes to cause them to behave in arbitrarily malicious ways. Typically, a significant engineering effort is required to corrupt a process so that its behavior will have malicious effects. Most likely, then, an attacker will automate this part of the attack off-line so that it can be applied quickly on-line.

In what way is the attacker limited? If the attacker can concurrently attack many domains, gaining privilege in each, corrupting application processes in each, even a Byzantine fault-tolerant group will soon fail. Perhaps successful attacks will be detected in some domains and those domains will be quarantined. But intrusion detection is far from perfect; we did not expect our defense to rely solely on it.

We assumed that the attacker cannot gain privileges in many domains at once. With this assumption, even if the attacker attacks many domains concurrently, those attacks will succeed at different times, leading to corruption of the system in stages. We called this the *staged attack assumption.* Given this assumption, and even with imperfect intrusion detection, the defense may have time to react to an attack, delaying the attacker's eventual success or possibly quarantining the attack completely.

The staged attack assumption is plausible if security domains differ. As explained in Section 5.1, if domains are configured differently, an attack that works against one may not work against another. Then the attacker must spend time trying to find an attack that will work, and we expect that this trial and error process will take different amounts of time for different domains.

The Validation Report contains several models of the interaction between the attacker we have just described and the ITUA defenses. These models differ in the details, but they all have the same basic outline. The models also have parameters to describe the attacker: how fast do attacks proceed? how good is intrusion detection? and so on. Whereas this section has described the model of the attacker in rough terms, the models in the Validation Report are much more precise, allowing mathematical analysis.

We summarize here a few issues we faced in developing the models in the Validation Report. Section 5.9 continues the discussion of models with some issues that arose during their analysis.

- **Should defenses focus on the worst case or on typical attacks?**

  We would like the ITUA defenses to be effective in both cases. At one extreme, we want strong resistance to "ankle biters" who seek to probe the defenses but have done little preparation. At the other extreme, we also want to "raise the bar" against well-prepared attackers, even if their most sophisticated attacks will eventually overwhelm our defenses.

  Some of our defense mechanisms, such as Byzantine fault-tolerant groups, work in both cases. Mechanisms that block or quarantine domains or hosts, however, typically force a trade-off between the extremes. If a naive attacker can probe our defenses, causing us to block or quarantine, a sophisticated attacker will probably avoid such probing and instead try to spoof the defenses to cause blocking or quarantine of the wrong domain, thus denying service.

When we have been forced to choose, we tended to design ITUA for the worst-case attacks.

In the Validation Report, Chapters 2, 4, and 5, our analysis focused on the worst case, in which the attacker infiltrates domains as quickly as possible, corrupting processes instantaneously and creating as much damage as possible. In the Validation Report, Chapter 3, we modeled the attacker more generally, allowing an analysis of a more typical case. In that model, infiltration and corruption spread through the system at rates that are constant per unit time. In that model, the attacker's progress toward gaining privilege is partly governed by chance rather than always taking the shortest time.

- **Is diversity necessary to cause staged attacks?**

  If the attacker is unprepared, diversity is not necessary. The unprepared attacker may spend significant time exploring the system using trial and error, thus collecting system privileges in stages.

  In the worst case, however, the attacker will be prepared. If the security domains are homogeneous and if any domain can be attacked from any other, the attacker will apply the same attack to every domain, killing the system.

  One possible way to force staged attacks would be to restrict network communication so that not all pairs of domains can talk to each other. Then an attacker in one domain could not arbitrarily attack any other. Although this approach might work, it is highly undesirable because it works against network transparency.

  We know of no good mechanism other than diversity for enforcing the staged attack assumption.

- **Should flooding attacks be considered?**

  Flooding attacks are a significant problem. In the ITUA project, however, we treated them as out-of-scope. Therefore, the models in the Validation Report assume that the attacker does not use flooding.

5.3 Global Coordination

The effectiveness of defense in the ITUA architecture depends on coordination between the ITUA managers. In Section 5.1, we described *how* the managers will coordinate. In this section we describe *what* the managers should coordinate to do. We list the kinds of decision that must be coordinated and discuss how best to make those decisions. In other words, ITUA and systems like it need a *policy* for coordinating their defense. We have found, not surprisingly, that some policies are better than others.

ITUA managers must coordinate two kinds of decision:

1. when and how to reconfigure groups of replicas;

2. when to quarantine security domains.

We addressed the following key questions for coordination policy:

- **What is the best size for a replica group?**

  The larger the group, the more Byzantine corruption it can tolerate. Byzantine fault-tolerant protocols of the kind we use in every replication group will tolerate Byzantine corruption of any number of replicas less than a third of the group size. So a group with 4 replicas will tolerate corruption in one replica, a group with 7 replicas will tolerate corruption in at most two replicas, and so on. Clearly, larger groups provide better tolerance.

  On the other hand, larger groups have costs. The first cost is of the resources needed to run the replicas. To make it hard for the attacker to corrupt many replicas at once, we run each replica in a different security domain. Therefore larger groups need more security domains and, as already noted in Section 5.1, creating each new security domain is relatively expensive. The second cost is the overhead of the Byzantine fault-tolerant protocol needed to coordinate the replicas. This cost also grows larger for larger groups.

  In the Validation Report, Chapter 2, we suppose that number of security domains is unbounded and concentrate on the trade-off between tolerance and overhead. To optimize this trade-off, we define a measure of goodness, called "useful life", that tells how much computation the defense-enabled application can expect to get before the worst-case attack overwhelms the defense. Forcing the attacker to corrupt more domains will tend to increase useful life but the overhead will tend to decrease it. We make some reasonable assumptions about the overhead, in particular, that it scales quadratically with the group size, then we show mathematically that:

  - Tolerating 1 corrupt replica, i.e., a group size of 4, gives more useful life than not tolerating corruption at all, i.e., a group size of 1.

  - Larger groups reduce the useful life. In other words, the increased overhead of the larger group is more expensive than the increased time the larger group can expect to survive.

We conclude that the ideal replication group size is 4.

- **Should group size be increased under attack?**

  Although the ideal group size is 4, perhaps larger group sizes with their larger overhead are better when the group is under attack. In that case, perhaps the increase in tolerance is more beneficial than the extra computation lost to overhead.

  In the Validation Report, Chapter 2, Section 6, we compare a policy of pure replacement with a policy that changes the group size from 4 to 7 after an attack is detected. Both policies maintain at least 4 replicas in the group at all times by discarding any replica in a domain that has been detected to be corrupt and replacing it with another in a good domain, but the new policy adds three replicas at the first sign of trouble.

  Mathematical analysis showed that the useful life is greater under the policy of pure replacement unless the time to reconfigure the group, (i.e., to change its membership) is very long, in which case the alternative policy offered a slight advantage. Because we aimed to keep the cost of reconfiguration as small as possible (for reasons in addition to survivability) we chose the pure replacement policy.

- **When should a group be reconfigured?**

  At a minimum, the group must be reconfigured when one of its replicas is running in a security domain that has just been quarantined for corrupt or suspicious behavior. Then that replica is cut off from the rest of the group. To maintain 4 replicas, it must be replaced by another. Similarly, reconfiguration is forced if a replica simply crashes, even if not under attack.

  But should the group be reconfigured in other circumstances too? We did not completely answer this question during the ITUA project. It may be prudent to move replicas away from domains that are suspect but not quarantined. On the other hand, reconfiguring too often would waste resources and may give the attacker ways to trick the defense into wasting them.

  Our approach to answering the question for ITUA was to reconfigure only when forced to, by crashes or quarantine, but also to quarantine according to a policy that optimizes survivability. Such policies are addressed next.

- **Should demonstrably corrupt security domains be quarantined?**

  Yes. If the manager in domain D is certain that domain X is corrupt, D should block X. Additionally, if the manager in D can prove to other managers that X is corrupt, it should forward that proof and X should be quarantined by all correct managers.

- **Should suspicions be used to quarantine security domains?**

  In general, no, because of the risk of spoofing and denial of service. The decision must also depend, though, on a judgement about the likelihood that the suspicion is correct.

  The Validation Report, Chapter 5, analyzes one common special case in detail. If D is certain that X is corrupt but cannot prove it to other managers, then D's claim "X is corrupt" is only a suspicion for those other managers, because D may itself be corrupt and lying. The suspicion, however, carries a lot of information, because at least one of D or X *is* corrupt.

  Chapter 5 shows that, if the number of domains is much larger than the size of a replication group (usually 4), the defense should quarantine both D and X to maximize the probability that the attack is contained and thereby maximize the survival of the application. Our analysis of the best policy if the number of domains is small was not completed during the ITUA project.

5.4 Rapid Local Response

The ITUA design includes rapid local response to events that may indicate an attack is in progress. These responses are local to each host and are much faster than the global adaptive responses described in the previous section because no coordination among the managers is involved. A faster response means that the chance of delaying or foiling an attack is greater.

A control loop implements each rapid local response. The control loop monitors one or more sensors, each of which is typically a 3rd party Intrusion Detection System (IDS). From the sensor data and possibly other state, a decision is made and changes to the local host are made via one or more actuators. Sensors we have used include:

- the Snort network IDS [14];
- the Tripwire file integrity checker [9];
- a homegrown detector for TCP connection floods, which counts the number of existing connections on each port and raises an alarm if the number is too high;
- a homegrown detector for ARP cache poisoning, which remembers the state of the ARP cache and raises an alarm when it is changed.

Actuators we have used include:

- the Netfilter Linux firewall [11];
- homegrown scripts for checkpointing and restoring data;
- host shutdown.

As with global coordination, rapid local response raises various policy issues:

- **How aggressive should local responses be?**

  The answer depends on a judgement about the reliability of the sensors and of the meaning of the sensor data. IDSs are notoriously imprecise: actual attacks are missed and false alarms are raised. The attacker may also be able to spoof, or fool the detector into raising the wrong kind of alarm.

  For example, an attacker may use a port scan as the first step in an attack, to find which ports a host has open. A sensor that reports port scans could easily report which host was the source of the scanning, so that that host could be blocked. Unfortunately, a port scan detector is easily spoofed so that it reports the wrong source. Blocking a host based on such port scan spoofing would deny service to authorized uses.

  In general, an aggressive response is good if it has a high probability of stopping unauthorized behavior. An aggressive response is bad if it is easily spoofed into stopping authorized behavior instead.

- **Should local responses involve the manager on the local host?**

  Typically, yes. The manager serves two main functions in a rapid response control loop. First, it can set policy that applies to every actuator. If aggressive responses are dangerous because resources in the network are scarce and denial of service is a big risk, the managers can agree to be less aggressive and cause every local control loop to respond less aggressively. Second, some control loops keep state and the manager can be a good place to maintain that state.

5.5 Byzantine Group Communication

The ITUA architecture puts processes with redundant functions into a group. Specifically, every replicated application component consists of a group of its replicas, and the set of all ITUA managers forms a single, cooperative, group.

All ITUA process groups have communication needs in common. The group members communicate via *multicast*, in which one member sends the same message to every other. The multicast protocols must be able to ensure that every correct group member receives the same messages in the same order, in spite of Byzantine corruption of some members. The group must also be able to agree on its membership, i.e., which processes should receive a multicast and which should not, and it must be able to cope with changes in membership forced by corruption and by decisions of the ITUA managers.

ITUA handles these needs with a Group Communication System (GCS) used by every ITUA group. The GCS implements three main protocols for:

1. multicasting messages reliably;
2. putting multicast messages into a total order;
3. maintaining membership data.

The GCS also supports the transfer of state from one replica to another; this state transfer is used in replica groups when starting a new replica but is not needed by the manager group. Each GCS protocol tolerates intrusions and failures by guaranteeing correct behavior if fewer than a third of the group members are corrupt. More detail about the GCS design can be found in [13].

The following key questions arose during development of the GCS:

- **Is the intrusion tolerance required of the GCS fundamentally different from the Byzantine fault tolerance required of other systems?**

  The requirements placed on the ITUA GCS are more severe than the requirements placed on some, and perhaps most, Byzantine fault-tolerant systems. We first discuss how the requirements are similar, then discuss how intrusion tolerance goes beyond fault tolerance.

  Intrusion tolerance is like Byzantine fault tolerance in that corruption of a subset of group members must be tolerated. In both cases we aim to tolerate Byzantine corruption, which is the weakest assumption that can be made about the behavior of a process. In both cases we want reliable multicast protocols that preserve the order in which messages are delivered.

  Intrusion tolerance is more demanding than Byzantine fault tolerance because it must work even in a very dynamic environment. The ITUA GCS must implement state transfer to new replicas because replicas will certainly be killed or corrupted by an attacker, whereas this capability may not be necessary for some fault-tolerant systems if failures are rare. The ITUA GCS protocols must work even when the group membership changes frequently. For example, when the ITUA protocols use voting, care must be taken to maintain a record of votes cast even when the group membership changes.

Thus, the protocol can guarantee the progress of voting in spite of an attack that forces the group membership to change repeatedly during the vote.

- **What model of network communication should the GCS assume?**

  One way that fault-tolerant group communication protocols differ is in the assumptions they make about the underlying network. A distinction is often made between *synchronous* and *asynchronous*: in an asynchronous network, every message will be delivered eventually (perhaps using retransmissions), whereas a synchronous network delivers within a bounded time. The assumption made about a synchronous network is clearly stronger than the assumption made about an asynchronous one.

  Stronger assumptions tend to be worse for intrusion-tolerant systems because they allow more opportunity for attack. So protocols that assume a synchronous network might be made to fail if the attacker is able to delay messages. Protocols that assume only an asynchronous network are better because they will not fail under such attacks.

  On the other hand, asynchronous networks force two limitations on GCS:

  1. A deterministic protocol for agreement is impossible [7]. Agreement protocols are necessary for the manager coordination of Section 5.1 and they underlie the reliable, ordered multicast GCS protocols. If the network is asynchronous, the GCS cannot be deterministic.
  2. Progress of the GCS protocol and of the services that depend on it cannot be guaranteed in any finite time. As a practical matter, this limitation is never acceptable in real systems because services that take too long are assumed to be broken, whether they really are or not.

  To avoid these limitations, we assumed a *timed asynchronous* network [5]. This assumption is intermediate between synchronous and asynchronous: most messages are delivered within a bounded time, T, but exceptions are possible, and the frequency of exceptions is also bounded. This assumption approximates real networks of workstations in ordinary use fairly well [2].

  Choosing the parameter T turned out not to be an easy task in ITUA. Too large, and the system waits too long to time out in response to real failures. Too small, and the system raises false alarms about components that are slow rather than dead.

[2]We realize, though, that it may not approximate a network under attack.

5.6 Unpredictability

We claim that defensive adaptation is more effective if it is unpredictable. To test this claim, the ITUA project explored ways in which unpredictability can be added to other defense mechanisms. Most of our work centered on creating a general method for adding unpredictable alternatives rather than on implementing the specific alternatives themselves.

Unpredictability can be added to almost any adaptive response by making a *random* selection among the available alternatives. If needed, fake or decoy alternatives can usually be created. The key questions we addressed about this approach are the following:

- **Is a general mechanism for creating randomness in application-level adaptation possible?**

  Yes. Many of the application-level adaptation decisions that ITUA implements are made in middleware. We used the QuO adaptive middleware [10] as a framework for describing these decisions and how they should be made. When a choice among alternatives is possible, an alternative can be chosen randomly. We modified the QuO framework to support such random choices.

- **What adaptation decisions offer alternatives that can be made randomly?**

  We identified these options:

  - If a new replica is needed in some group, in which security domain and on which host should it be created?
  - If a security domain contains hosts or processes that are behaving in a suspicious manner but are not yet provably corrupt, should the domain be quarantined?
  - If an attack is detected using a (TCP or UDP) port P and the attacking host cannot be reliably identified, blocked or quarantined, should P be closed and for how long? (See the Validation Report, Chapter 4 for an analysis of some of these options.)
  - If the defense cannot maintain full replication of a component and must enter a degraded mode of operation, should it create decoys that look like component replicas?
  - Should normal communication include fake messages interleaved with real ones to confuse traffic analysis?

- **Does random choice between alternatives guarantee unpredictability?**

  The answer depends on where the observer is. To an outside observer, e.g., an attacker who has no privilege in the system, a random choice between alternatives will appear unpredictable. This is true even when the number of alternatives is small. However, if the observer is inside the entity that makes the random choice then the observer will be able to predict the behavior of the system following the choice. So the general answer is that unpredictability cannot be guaranteed, but a random choice will be unpredictable to some observers.

  Consider the worst case for the defense. A decision, such as where to create a new replica, will be made by a set of managers, including at least one manager from every domain that is not quarantined. If the attacker has silently corrupted one of the managers in that set, he will learn the result of the decision before that decision is put into effect. Therefore, a random decision will be predictable in this case.

  Two factors affect the likelihood an attacker will find random behavior predictable. First, the more distributed the decision-making, the more likely it is that the attacker has already corrupted some component that will tell him the result of the decision. Second, the more components the attacker has corrupted within the system, the more likely it becomes that he can collect the information he needs to find out which random choice was made.

  Therefore, a random choice is more likely to be unpredictable if it is made by as few components as possible and if the defense aggressively quarantines domains thought to be corrupt, cutting them out of the system.

5.7 Technology Transition

To test how easily ITUA could be applied to a system of military interest, we began with IEIST [4] (Insertion of Embedded Infosphere Support Technologies), a C2 system with both ground nodes and aircraft. IEIST was concurrently under development at Boeing and shared several developers with the ITUA project. Taking several IEIST components, we created an application that executes a basic IEIST scenario over a network of hosts (without the need for actual aircraft), and defense-enabled it.

IEIST focuses on the development of software called Guardian Agents (GAs) and Fuselets. A GA augments an embedded tactical system, interoperates with legacy systems and communication links, and plugs into the Joint Battlespace Infosphere (JBI) [18]. A Fuselet is an entity in a JBI whose job is to derive new knowledge by *fusing* multiple observations. A typical IEIST use-case has interactions among a fighter-aircraft GA, an Unmanned Combat Air Vehicle (UCAV) GA, and a Discovery and Navigation Fuselet. To make UCAV data available to the fighter, both the UCAV and fighter GAs register with the Fuselet and the fighter GA announces its route plan. The Fuselet connects the fighter GA with UCAV GA(s) that are monitoring the fighter's route and have relevant information. The Fuselet serves multiple fighters and multiple UCAVs simultaneously and is therefore a critical component of this application.

Our defense enabling centered on surviving attacks against the Fuselet. We replicated the Fuselet in 4 security domains to tolerate Byzantine corruption of one domain. Three additional domains were used, two to run stand-alone non-replicated versions of the Fuselet and a third to run the GAs. The defense strategy was to maintain 4 Fuselet replicas, abandoning domains for which severe intrusion-detection alerts had been raised, and switching to using the stand-alone Fuselets if fewer than 4 domains remain available.

Applying ITUA to IEIST addressed the following issues:

- **Were the ITUA defenses effective for this application?**

  The ITUA defenses worked as designed. An attacker who infiltrates one domain and is detected will cause that domain to be abandoned and its Fuselet replica started in another domain. The ITUA defense switched to using stand-alone Fuselets under the expected conditions.

  However, the ITUA project did not measure the difficulty for an attacker to carry out the above attack, nor did it measure, e.g., in Red Team experiments, the difficulty for an attacker to circumvent the ITUA defenses. Within the ITUA team, the defenses received a great deal of scrutiny, including some of the analysis and planning that would go into a Red Team experiment. We believe, without proof, that the ITUA defenses would add significantly to the difficulty of an attack, even by a sophisticated attacker.

- **Were changes to IEIST needed to use ITUA?**

  No changes to IEIST functionality were needed. To replicate the Fuselet, however, some Fuselet-specific code had to be written and one problem with nondeterminism needed a workaround. We consider this nondeterminism problem to be a significant lesson learned from ITUA.

  The Fuselet-specific code was needed for transferring the state of one replica to another. The code allows a snapshot to be taken of the state of one replica, and allows that snapshot to initialize the state of a new replica when it is started. Such application-specific code is required in any system that can dynamically change the membership of a replication group that maintains an internal state.

Using ITUA also requires that replicas behave deterministically[3]. If each replica receives the same inputs in the same order, then determinism implies that the replicas will maintain the same state, i.e., they will continue to be replicas. The IEIST Fuselet is deterministic, and therefore can be replicated using ITUA.

However, the middleware that ITUA uses for communication (the TAO [16] implementation of CORBA) does not marshal data deterministically. Therefore, application data sent by two replicas may be transmitted over the network in two different forms. Unfortunately, ITUA compares such data at a low network layer where this difference matters. Our workaround for this problem was to avoid all cases of nondeterminism in CORBA and this workaround required changes to IEIST. A better solution would have been to compare data from different replicas at a higher layer at which this nondeterminism never appears.

- **Would the ITUA defense mechanisms be practical in this application domain?**

  The ITUA *approach* to defense enabling appears to be practical and desirable for this domain. However, this domain includes real-time applications. We suspect that the existing ITUA *implementation* would not provide the real-time predictability that is needed for such applications.

5.8 Implementation

We implemented a prototype of ITUA, including global coordination via managers (Section 5.3), local control loops for rapid response (Section 5.4), and a supporting Byzantine fault-tolerant group communication system (Section 5.5). Following are some key implementation issues that surfaced:

- **How should concurrency within a manager be controlled?**

  Every ITUA manager must respond to a variety of inputs, including notifications from sensors on the local host and multicasts from other managers. These inputs represent concurrent activity of the system. Most of these inputs can cause changes to the internal state of the manager and therefore

[3]For active replication. It is to be noted that other replication schemes are also possible under ITUA where this deterministic behavior requirement on the replicas can be relaxed.

the order in which the manager processes the inputs may affect whether the manager's state, and thus its behavior, is correct.

Initially, we gave each input its own thread, allowing the manager to process inputs concurrently. As more functionality was added, though, synchronizing access by these threads to the manager's data became increasingly difficult. Eventually we abandoned this approach as too complicated.

Some of the inputs are remote procedure calls that expect a reply. We simplified the manager design by serializing these calls, forcing each to run to completion before handling the next. Other inputs are still handled concurrently with these calls.

One problem with this new approach was that waiting for completion of some calls may take a long time, and other calls, that wait on receipt of inputs from managers that may have failed, may take forever. We solved this problem by implementing a timer queue that allows controlled interruption of the remote procedure call currently in progress.

- **How should consensus decisions by the managers be arranged?**

  Each decision is triggered by one or more managers multicasting a request to begin the decision-making process. If the managers agree that a decision is needed, then they make the decision.

  Our original design had the managers, after agreeing to make a decision, multicast their current version of the data on which the decision would depend. Once every correct manager has every other manager's current data, a consensus decision can be made by applying a deterministic function to the data. The drawback of this design was that it took two separate agreement steps to reach a decision, and that was slow.

  We changed the manager design so that data on which decisions would depend are multicast whenever the data change. This, in effect, replicates that data at each manager because every manager always knows the state of every other manager's data. The effect of this change was to keep the managers perpetually ready to make a decision, reducing a two-step process to the one step of agreeing on when a decision needs to be made.

  The only drawback of this design change is that multicasting several versions of the same data between decisions is wasteful. For ITUA, paying this price was a small cost for the gain of quicker decision-making in the manager group.

5.9 Validation

Section 5.2 discussed the ITUA model of the attacker. In the Validation Report, that model was combined with a model of the ITUA defense and its environment in order to estimate the survivability provided by ITUA. This section discusses some issues that arose in making that survivability estimate.

- **Is a stochastic model necessary?**

  Yes. Probabilities enter the analysis in several ways. Some mechanisms that depend on factors too detailed to appear in the model may be modeled as stochastic processes. The most important example is intrusion detection. At the level of abstraction in our models, we cannot know whether a particular attack will be detected, and we do not even know whether the detector is deterministic, i.e., that if an attack is detected once that it will be detected every time. Because of this uncertainty, we modeled intrusion detection as a stochastic process parameterized by several probabilities and rates.

  Probabilities will also enter the analysis if the attack or the defense uses an unpredictable strategy. Such strategies arise, for example, in the model of quarantine in Chapter 5 of the Validation Report.

- **Is game-theoretic analysis necessary?**

  In general, yes. The interaction between attack and defense is a game. The defense plays the game to maximize some measure of survival against the best attack. Likewise, the attacker plays to minimize survival against the best defense.

  We made no use of game-theoretic methods in Chapters 2, 3, and 4 of the Validation Report because the best strategy for the attacker was clear: attack as aggressively as possible. Therefore the analysis of these models could be simplified.

- **How were model parameters estimated?**

  Our plan was to study actual systems and actual intrusions to estimate these parameters. That plan was only partly completed during the ITUA project.

  We built a tool, Ferret, for finding vulnerabilities in systems. From the frequency at which vulnerabilities are found in real systems, we estimated how likely an attacker would be to exploit the vulnerabilities during an actual attack, and from this, we estimated parameters of the model that measure how quickly security domains will be overrun in an aggressive attack. The Validation Report, Chapter 6, describes Ferret and describes our approach to parameter estimation, working out an example in detail.

6 ITUA Deliverables and Outreach

6.1 Technology Demonstrations

We demonstrated ITUA technology on the following occasions.

Demonstrating Intrusion Tolerance with ITUA, DISCEX III, Washington, DC, April 2003.

SAMS demonstration including ITUA technology at CECOM, Ft. Monmouth, April 2003.

ITUA Demonstration at the OASIS Winter 2002 PI meeting, Hilton Head Island, SC, March 2002.

ITUA Demonstration at DARPA Tech 2002, Anaheim, CA, July 2002.

ITUA Demonstration at the OASIS Summer 2001 PI meeting, Santa Fe, NM, July 2001.

6.2 Project Software

The events above demonstrated software prototypes resulted from the ITUA project. These include new software developed or enhancement and integration of existing software by the ITUA team as described below.

Development of the ITUA Technology suite consisting of rapid reaction loops, intrusion tolerant gateway and GCS and enhanced version of QuO contracts.

Development of Ferret, which is a host vulnerability checking tool.

Defense enabled IEIST application, which demonstrates the integrated capability of the ITUA technology suite in the context of the Discovery and Navigation Fuselet and the Guardians in Boeing's IEIST application.

QuO and Gateway Components for SAMS, that were integrated with other OASIS technologies to defend the security monitoring service.

6.3 Published Papers

We wrote the following papers about ITUA and ITUA-related topics. The first three were included in the OASIS book (*OASIS- Foundations of Intrusion Tolerant Systems*, edited by Jay Lala, published by DARPA and IEEE Computer Society Press.

Survival by Defense-Enabling. P. Pal, F. Webber, R.E. Schantz, J. Loyall, R. Watro, W. Sanders, M. Cukier, and J. Gossett. Proceedings of the New Security Paradigms Workshop 2001, Cloudcroft, New Mexico, September 11-13, 2001, pp. 71-78.

Quantifying the Cost of Providing Intrusion Tolerance in Group Communication Systems. H. V. Ramasamy, P. Pandey, J. Lyons, M. Cukier, and W. H. Sanders. Proceedings of the 2002 International Conference on Dependable Systems and Networks (DSN-2002), Washington, DC, June 23-26, 2002.

Formal Specification and Verification of a Group Membership Protocol for an Intrusion-Tolerant Group Communication System. H. V. Ramasamy, M. Cukier, and W. H. Sanders. Proceedings of the 2002 Pacific Rim International Symposium on Dependable Computing (PRDC 2002), Tsukuba, Japan, December 16-18, 2002.

An Architecture for Intrusion Tolerance Using Adaptation and Redundancy. Partha Pal, Paul Rubel, Michael Atighetchi, Franklin Webber, William H. Sanders, Mouna Seri, HariGovind Ramasamy, James Lyons, Tod Courtney, Adnan Agbaria, Michel Cukier, Jeanna Gossett, Idit Keidar. Submitted for publication.

Semi-Passive Replication in the Presence of Byzantine Faults. HariGovind V. Ramasamy, Adnan Agbaria and William H. Sanders. Submitted for publication.

A Byzantine Agreement Protocol with Fault-Proportionate Overhead. H. V. Ramasamy, A. Agbaria, and W. H. Sanders. Submitted for publication.

Ferret: A Host Vulnerability Checking Tool. A. Sharma, J. R. Martin, N. Anand, M. Cukier, and W. H. Sanders. Proceedings of the IEEE Pacific Rim International Symposium on Dependable Computing (PRDC-10), Tahiti, French Polynesia, March 3-5, 2004.

Formal Verification of an Intrusion-Tolerant Group Membership Protocol. H. V. Ramasamy, M. Cukier, and W. H. Sanders. IEICE Transactions on Information and Systems special issue on Dependable Computing, vol. E86-D, no. 12, December 2003.

Dependability and Performance Evaluation of Intrusion-Tolerant Server Architectures. V. Gupta, V. Lam, H.V. Ramasamy, W.H. Sanders, and S. Singh. Proceedings of LADC 2003: The 1st Latin American Symposium on Dependable Computing, São Paulo, Brazil, October 21-24, 2003.

Probabilistic Validation of an Intrusion-Tolerant Replication System. S. Singh, M. Cukier, and W. H. Sanders. Proceedings of the 2003 International Conference on Dependable Systems and Networks (DSN-2003), San Francisco, CA, June 22-25, 2003, pp. 615-624.

Providing Intrusion Tolerance With ITUA. T. Courtney, J. Lyons, H. V. Ramasamy, W. H. Sanders, M. Seri, M. Atighetchi, P. Rubel, C. Jones, F. Webber, P. Pal. R. Watro, Michel Cukier and J. Gossett. Supplemental Volume of the 2002 International Conference on Dependable Systems and Networks, June 23-26, 2002.

Probabilistic Validation of Intrusion Tolerance. W. H. Sanders, M. Cukier, F. Webber, P. Pal, and R. Watro. Digest of Fast Abstracts: The International Conference on Dependable Systems and Networks, Bethesda, Maryland, June 2002.

A Configurable CORBA Gateway for Providing Adaptable System Properties. M. Seri, T. Courtney, M. Cukier, V. Gupta, S. Krishnamurthy, J. Lyons, H. Ramasamy, J. Ren, and W. H. Sanders. Proceedings of the Workshop on Dependable Middleware-Based Systems (WDMS 2002), Washington, DC, June 26, 2002.

Intrusion Tolerance Approaches in ITUA. M. Cukier, J. Lyons, P. Pandey, H. V. Ramasamy, W. H. Sanders, P. Pal, F. Webber, R. Schantz, J. Loyall, R. Watro, M. Atighetchi, and J. Gossett. FastAbstract in Supplement of the 2001 International Conference on Dependable Systems and Networks, Göteborg, Sweden, July 1-4, 2001, pp. B-64 to B-65.

Assessing Adaptation in the Context of Security and Survivability. P. Rubel and P. Pal. Presented as a position paper in the First Workshop on Information-Security-System Rating and Ranking (ISSRR), Williamsburg, VA, May 2001.

Impediments to Building Survivable Systems: An Experience Report. P. Rubel, P. Pal, F. Webber, M. Atighetchi, and C. Jones. IEEE ISW 2001/2002.

Intrusion Tolerant Systems. P. Pal, F. Webber, R. Schantz, and J. Loyall. Proceedings of the IEEE Information Survivability Workshop (ISW-2000), 24-26 October 2000, Boston, Massachusetts.

6.4 Theses

Students under the direction of Bill Sanders at the University of Illinois produced the following theses related to ITUA.

Intrusion-Tolerant State Transfer for Group Communication Systems. V. Gupta. Master's Thesis, University of Illinois, 2003.

Probabilistic Validation of an Intrusion-Tolerant Replication System. S. Singh. Master's Thesis, University of Illinois, 2003.

A Replication Protocol for an Intrusion-Tolerant System Design. J.P. Lyons. Master's Thesis, University of Illinois, 2003.

An Adaptive Quality of Service Aware Middleware for Replicated Services. S. Krishnamurthy. Ph.D. Thesis, University of Illinois, 2002.

A Group Membership Protocol for an Intrusion-Tolerant Group Communication System. H. V. Ramasamy. Master's Thesis, University of Illinois, 2002.

Reliable Delivery and Ordering Mechanisms for an Intrusion-Tolerant Group Communication System. P. Pandey. Master's Thesis, University of Illinois, 2001.

6.5 Other ITUA Presentations

Several other occasions called for ITUA-related talks.

Bill Sanders represented our project in the *joint EU-US workshop on Intrusion and Attack Tolerance* (January 29-30, 2001, Lisbon).

Partha Pal presented our work-in-progress at the *IEEE ISW-2000 workshop* (October 24-26, 2000, Boston).

Franklin Webber was on a *panel on "Integrating Fault Tolerance and Security in Distributed Information Systems"* at the 19th IEEE Symposium on Reliable Distributed Systems (October 16-18, 2000, Nuremberg, Germany).

6.6 Project Documents

The following documents were produced as project deliverables.

Performance and Red Team Evaluation Report

Validating Intrusion Tolerance Using Mathematical Models

Validation and Characterization Report

Final Software Release Users' Guide

ITUA Gateway Reference Manual

OASIS Peer-Review Document

Finally, the following technical reports also document results of research performed under this project.

Overcoming Byzantine Failures using Checkpointing. Adnan Agbaria and Roy Friedman. University of Illinois at Urbana-Champaign Coordinated Science Laboratory technical report no. UILU-ENG-03-2228 (CRHC-03-14), December 2003.

Stochastic Modeling of Intrusion-Tolerant Server Architectures for Dependability and Performance Evaluation. V. Gupta, V. Lam, H. V. Ramasamy, W. H. Sanders, and S. Singh. University of Illinois at Urbana-Champaign Coordinated Science Laboratory technical report no. UILU-ENG-03-2227 (CRHC-03-13), December 2003.

7 Plans for Further Research

The results obtained in ITUA reinforce the need for continued further exploration of a number of topics, that can be roughly identified as falling within the three broad areas:

- **Integration of available Intrusion Tolerance solutions:** ITUA and other OASIS projects provided solutions to specific problems. These individual technologies need to be put together in an integrated solution to understand their collective capabilities.

- **Further exploration of open issues and issues uncovered during the present phase of research:** Exploration to date both within the outside ITUA, clearly shows that several key issues in intrusion tolerance are still open (some of these are discussed later in this section). Without satisfactorily addressing these issues, integration of multiple technologies will remain weak, and vulnerable.

- **Software Engineering of survivable systems:** Even though the individual technologies have started to emerge and mature, the technology integration and software engineering of intrusion tolerant systems are still very human-centered activity. This impedes mainstream adoption of intrusion tolerance technology in producing repeatable solutions in an economic way.

Continued work in all these areas is needed to develop a solid, transitionable foundation of intrusion tolerance techniques on which future survivable systems can be built. In this section, we briefly describe a selected sample of research problems and issues within each of these areas.

7.1 Integrated Survivability Solutions

The need for technology integration was visible very early within the OASIS program. The effort to characterize the technologies being produced by various projects in the OASIS program clearly showed that none of the individual projects could achieve comprehensive survivability for a typical DoD applications. Even though many projects, including ITUA, explored an architectural approach, they fell short of a generic framework that could be customized for specific cost-survivability trade-offs of individual projects. Most of the technologies being explored in OASIS and prior DARPA programs investigating 3rd generation security addressed specific problems, and had specific target applications.

We have been espousing an application oriented approach to survivability (APOD), that started with understanding the survivability requirements of an application, followed by devising a survivability strategy that best serves those requirements, and subsequent realization of that strategy in the system. We carried forward this approach to utilize the ITUA technologies in defending applications like IEIST, where corruption inducing attacks are more attractive to the adversary than any other kind of attacks. Nevertheless, the big picture of a survivability architecture, where multiple diverse technologies are integrated to provide a comprehensive solution was still lacking.

The SAMS project within OASIS was a small-scale experiment in that direction. While it established that multiple OASIS technologies could be integrated to achieve something useful, it produced a very customized, one of a kind solution. The real exploration of a comprehensive security architecture has began, and is continuing in the OASIS Dem/Val program. We believe that results from OASIS Dem/Val will provide a proven direction for building survivable systems in the near future, and will help establishing new agendas for survivability research in the long run. Coordinated use of protection, detection and reaction capabilities;

design principles for survival; quantifiable system agility (how quickly can the system adapt?) and resiliency (can the system provide some service even if major parts of the system are unusable?) are important research goals in this context. Some of these are being addressed in OASIS Dem/Val.

7.2 Continued Exploration of Survivability Technologies

In this section we outline some of the open research problems in Intrusion Tolerance.

- **Dependence on Timing and Environmental Assumptions:** Timing and environmental assumptions refer to things that an intrusion tolerance technology expects to be true. Strong assumptions create an unhealthy dependence that can be exploited by the attacker, and hence is a weakness (of the intrusion tolerance technology). This weakness may be ingrained at the basic concept of the technology or may stem from its implementation. In most cases, technology developers do not pay enough attention to this issue and its consequences, which is detrimental for the development and evolution of sound intrusion tolerance solutions. In the narrow context of ITUA, we made a conscious effort to use weaker assumptions (for instance our use of Timed Asynchrony as opposed to stronger synchronous assumptions). However, we observed that various key decisions are still time-dependent, i.e., depend on an event being timed out. As a consequence, the system is susceptible to attacker exploits of these time outs: by manipulating the system they can, in effect, steer the system to their desired mode of behavior or state. Further work is needed to address this issue. Better designed algorithms with less dependency on time outs as triggers, multiple layering of time out based behavior, and their containment and localization are examples of future research directions.

- **Graceful Degradation and Flexibility:** Algorithms and mechanisms that make static assumptions about their failure mode are inappropriate for survivable systems. On the extreme conservative side, systems can be built expecting the worst kind (I.e., Byzantine) of failures in the system. This is the approach taken in ITUA. However, Byzantine tolerant algorithms require that a certain level of replication be maintained (tolerance of f failures require 3f+1 replicas at least) and are also expensive in terms of communication costs. It may not always be practical to run the system with the goal of tolerating multiple Byzantine failures. Furthermore, if the f Byzantine failure tolerant system/protocol halts the system when the number of failure assumption is violated (i.e., more than f failures have occurred in a system of 3f+1 replicas), then the entire system fails to survive despite a significant level of resource remaining available in the system. It is often the case that the survivable system should continue to operate, even if it may not guarantee tolerance of further Byzantine failures. This calls for a flexible scheme where the defense can gracefully degrade i.e., going from tolerating say 1 byzantine failure to tolerating 1 crash failure when there are only 2 replicas available, and also upgrade i.e., going from 2 replicas with the ability to tolerate 1 crash failure to 4 replicas tolerating 1 Byzantine failure. We are exploring some of these ideas in our publish-subscribe protocols in DPASA.

- **Malicious Acts Within Privileges (Insider Attacks):** The ITUA architecture focuses on outsider attackers. The architecture does not include any defense against an insider user acting maliciously without violating his privileges. Consequently, such (insider) activities are considered outside the scope of our experiments. In order to extend the scope of our defense, we are proposing (elsewhere) to develop a technology that will introduce a high level of deterrence for the insiders to act maliciously. This deterrence will be built into the system by augmenting well-known insider mitigation techniques (such as double entry book keeping, split-authorization and auditing) with an on-line reasoning engine, which determines when to apply mitigating responses. To ensure the usability of the enhanced system (reducing the overhead of redundancy, computational checks and human intervention that are part of the mitigating responses), and to leverage uncertainty in the mind of the attacker, we will apply these responses in a probabilistic manner.

- **Furthering the Science of Validating and Quantifying Intrusion Tolerance:** We have developed an integrated methodology that combines different validation approaches such as testing, attack injection, and modeling. We applied this to evaluate significant parts of ITUA technology, and in the process identified a number of ways this methodology can be enhanced. One such extension consists of building an attack injection campaign representing realistic attack distribution. A second issue involves understanding and developing attacker profiles. In evaluating ITUA, some of the developed models needed an attack sub-model. Development of an approach to characterize the attacker will lead to more accurate and detailed attack sub-models, which in turn will improve the validation of intrusion tolerance. Furthermore, the attacker profiles will also help us estimate the various parameters in the sub-model better. This will make the model based evaluation more realistic.Another extension focuses on discovering all the vulnerabilities present in an intrusion tolerant system. Today, different tools exist to discover different kinds of vulnerabilities. In order to improve the validation of intrusion tolerance, these tools should be integrated and a more complete list of potential vulnerabilities should be looked for.

7.3 Better Software Engineering of Survivable Systems

- **Component Based Framework for Intrusion Tolerance(CoBFIT):** The ITUA experience has shown that it is often the case that key capability (e.g., event handling, distribution and replication over computing nodes, public-key cryptography, or intrusion detection) is needed in implementing multiple intrusion tolerance functionalities, protocols and algorithms. Yet it seems to be that each capability is custom built or integrated for different contexts. It is clear that by developing a reusable base of these key supporting capabilities, significant cost savings can be achieved in constructing and testing intrusion tolerance technologies. Toward that end, we have started working on a flexible software framework called CoBFIT, which stands for Component-Based Framework for Intrusion Tolerance. CoBFIT is intended to provide a robust, flexible, reconfigurable, and portable software framework for building intrusion-tolerant middleware services. The framework is meant to be extendable and adaptable to specific survivability and dependability needs, which cannot all be known in advance. A key challenge to achieving this goal is that we need to find a way to separate the common support for intrusion tolerance from intrusion-tolerant services, so that we have a reusable framework that could serve as a platform for building various intrusion- tolerant services without having to re-implement the required support for each of those services. To be successful, we will need to research ways to identify and separate key intrusion tolerance features from the the implementation of the intrusion tolerant service's functionality, as well as ways to organize the framework itself. Examples of such commonly needed features for supporting intrusion tolerance include secure buffer management, cryptography, coordination of inputs from multiple intrusion detection mechanisms, replication over multiple nodes for increasing dependability through redundancy, primitives for secure communication, and Byzantine consensus.

- **Languages and Tools Support:** Now that we have seen a positive indication that the concept of defense-enabling is viable, we need to work on making this process more streamlined and repeatable. Having better means of specifying the defense-strategy and tactic, code-generators or other compiling tools to produce code that can be used in defense-enabling, libraries of packaged, reusable adaptive behavior and other aspects of defense are important future directions. In another context we are exploring componentization of adaptive behavior. If successful, this will lead us to componentized defensive adaptation, and a step towards the library of reusable defensive behavior. We are exploring model driven architectures of distributed systems in another research project, which can be leveraged in specifying important elements of a survivability architecture as well.

8 Conclusion

ITUA was a very successful project in multiple ways. It achieved its primary goal of developing a technology and architecture for tolerating corruption-inducing attacks. This research broadened our understanding of cyber-defense and produced technology and mechanisms that are at the foundation of intrusion tolerant systems. The technological advancement, and the feasibility of transferring the new concepts and technology to military systems were demonstrated on multiple occasions including invited demonstrations at DARPATech 2002 and DISCEX 2003. Apart from our demonstration involving Boeing's IEIST application components, we transfered pieces of ITUA technology to other projects such as SAMS and OASIS Dem/Val. Our work in ITUA had direct impact on follow up research programs such as SRS and OASIS Dem/Val. We seeded new research ideas such as a flexible component based framework for intrusion tolerant communication layer and the use of uncertainty in defense. These concepts are now being explored in our continued research as well as by other researchers. We did seminal work in the area of validating intrusion tolerance. We developed a methodology for probabilistic quantification of intrusion tolerance that make use of logical analysis as well as modeling. We applied this technique to evaluate pieces of ITUA technology and parts of the DPASA survivability architecture (developed under the OASIS Dem/Val program). The results of our research in the ITUA project is documented in numerous publications. It generated 4 MS thesis and multiple PhD research topics (all under the supervision of Professor William Sanders at the University of Illinois).

As demonstrated by our achievements, we made an impressive stride with ITUA, but we are still at the early stages of a long way toward practical, usable, validated and quantifiable survivability. ITUA provided a partial solution, addressing only part of a problem. In DPASA (under OASIS Dem/Val) we are exploring an integrated architecture that combines state of the art COTS and DARPA developed technologies providing protection, detection and adaptive response capabilities. If successful, DPASA will demonstrate a new high watermark in survivability, that can be validated by experimental and analytic methods, and configured on the basis of cost-benefit trade-offs.

While we began exploring a more complete integrated survivability solution in DPASA, there are other technical problems to follow through. As documented in our test and evaluation report, we observed several anomalies in stress testing, uncovered a few drawbacks in our adversarial testing, and identified several trade-off and tuning issues. All these needs further exploration. Addressing these issues will improve the ITUA technology.

Finally, there are fundamental problems that we did not investigate so far. Handling attackers with privilege trying to damage the system by mounting legitimate actions (insider attacks), understanding and exploiting dynamic and behavior based trust in defense, survivability of highly dynamic and/or mobile systems, understanding interference between different survivability mechanisms, better software engineering for defense-enabling are some examples. We plan to investigate these areas in our continuing and future research projects. For instance, in MOBIES, we are exploring model driven architecture, which in conjunction with a library of adaptive defense, automated testing and intrusion injection support, can lead to a more robust engineering of survivable systems. The continued work at the University of Illinois on COBFIT aims to develop a better engineered object-oriented communication framework for intrusion tolerance that can be customized for various application contexts. As another example, one of the objectives of the DARPA SRS program is to develop technologies for thwarting insider attacks. Another upcoming DARPA program seeks to develop technologies for using dynamic alteration of trust relationship for defending applications running on mobile ad-hoc networks.

References

[1] M. Barborak, M. Malek, and A. Dahbura. The consensus problem in fault-tolerant computing. *ACM Comp. Surv.*, 25(2), 1993.

[2] Bob Blakely. The emperor's old armor. In *New Security Paradigms Workshop*, pages 2–16, September 1996.

[3] CERT advisories. http://www.cert.org/advisories/.

[4] D. Corman, T. Herm, and C. Satterthwaite. Transforming legacy systems to obtain information superiority. In *6th ICCRTS, CCRP*, June 2001.

[5] F. Cristian and C. Fetzer. The timed asynchronous distributed system model. *IEEE Transactions on Parallel and Distributed Systems*, 10(6):642–657, June 1999.

[6] Michel Cukier et al. AQuA: An adaptive architecture that provides dependable distributed objects. In *IEEE Symp. Reliable Distributed Syst.*, pages 245–253, October 1998.

[7] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):372–382, April 1985.

[8] S. Kent. On the trail of intrusions into information systems. *IEEE Spectrum*, December 2000.

[9] Gene Kim and Eugene Spafford. The design and implementation of Tripwire: A filesystem integrity checker. In *Proc. 2nd ACM Conf. Computer and Communications Security*, pages 18–29, 1994.

[10] Joe P Loyall, Richard E Schantz, John A Zinky, and David Bakken. Specifying and measuring quality of service in distributed object systems. In *Proc. IEEE Int'l Symp. Object-Oriented Real-Time Distributed Comp.*, pages 43–52, April 1998. Kyoto, Japan.

[11] Netfilter: firewalling, NAT and packet mangling for Linux 2.4. http://www.netfilter.org.

[12] Partha Pal, Michael Atighetchi, Franklin Webber, Rick Schantz, and Chris Jones. Reflections on evaluating survivability: The APOD experiments. In *IEEE Int'l Symp. Network Comp. and Applications*, April 2003.

[13] Partha Pal et al. An architecture for intrusion tolerance using adaptation and redundancy. In *Int'l Conf. Dependable Syst. and Networks*, June 2004. submitted.

[14] M. Roesch. Snort - lightweight intrusion detection for networks. In *USENIX LISA: Thirteenth Systems Administration Conference*, pages 229–238, 1999.

[15] R. Schantz and D. Schmidt. Research advances in middleware for distributed systems. In *World Computer Congress*, August 2002.

[16] Douglas C. Schmidt, David L. Levine, and Sumedh Mungee. The design of the TAO real-time object request broker. *Elsevier Science Computer Communications*, 21(4), April 1998.

[17] Fred B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Comp. Surv.*, 22(4), December 1990.

[18] United States Air Force Scientific Advisory Board. *Report on Building the Joint Battlespace Infosphere*, 1999. Volume 1: Summary; SAB-TR-99-02.

[19] US Department of Defense. *Trusted Computer System Evaluation Criteria (Orange Book)*, December 1985. DoD 5200.28-STD.

[20] John Wensley et al. SIFT: Design and analysis of a fault-tolerant computer for aircraft control. *Proc. IEEE*, 66(10), October 1978.